Specification for a Real-Time Transport Protocol

Alfred C. Weaver, W. Timothy Strayer, Bert J. Dempsey

Computer Science Report No. TR-89-02
January 10, 1989

Specifications for a Real-Time Transport Protocol

Alfred C. Weaver, W. Timothy Strayer, Bert J. Dempsey

Department of Computer Science
Thornton Hall
University of Virginia
Charlottesville, Virginia 22903
(804) 924-7605
acw@virginia.edu, wts4x@virginia.edu, bjd7p@virginia.edu

ABSTRACT

Conventional standard transport protocols do not adequately support distributed real-time systems. An ideal protocol would ensure that at all times the communications subsystem is working on the task (message) of the most value to the overall system. In this paper we investigate how to develop a flexible and potent discrimination scheme based on *importance*, an abstract concept that measures the degree to which a task contributes to the overall system goal. We examine typical communication subsystems for what can and can not be expected from such a policy. Finally, we review our chain of reasoning in reaching a proposal for a scheme implementable within currently existing or developing transport protocols.

1. Introduction

The subject of real-time, both in operating systems and communications, has generated much literature and as many definitions as there are systems which claim to be real-time. As an attribute, real-time commonly refers to a systems ability to accomplish its goals in the presence of time constraints. These time constraints usually manifest themselves as deadlines, and separate real-time systems into hard and soft deadlines. A hard real-time system is one for which missing a single deadline is disastrous; a soft real-time system relaxes the constraint to say that missing a deadline degrades the system. As systems become more distributed, the ability for a communications subsystem to provide services which are responsive to real-time needs is more widely demanded.

There are two approaches to endowing a system with real-time capabilities: provide services to system resources which are completed so quickly that the resources never represent a point of contention; or provide the system with a manner of arbitrating between competing tasks for the system resources. The former method is by far the more popular — increasing clock speed has an immediate and tangible effect on a system. However, advances in sheer speed cannot always outpace demands on a system. Furthermore, as systems become more highly loaded, differentiating among tasks to promote the most important activities at the expense of the less important ones becomes essential. Therefore, tasks within the system are usually *prioritized* by some scheme so that these tasks may be ordered while competing for shared resources. A priority is a mechanism by which a task has a relative importance assigned to it for use during competition for resources. This importance, therefore, imposes a ranking among all tasks. The server then chooses the highest priority task each time it can make such a choice. Within a computing environment, this satisfies, at least nominally, the concerns of users that all tasks are not equal.

One of the most valuable resources within a distributed real-time system is the communications subsystem. Information transfer and process synchronization depends on the timely and reliable performance of this subsystem. The communications subsystem must therefore be responsive to the needs of the applications within the system without hindering these applications unnecessarily. By making the underlying network as fast as possible, hopefully much of the contention can be eliminated. Unfortunately, performance measurements of several implementations of ISO standard communications protocols indicate that communications subsystem and the interface to the subsystem are inefficient and use a fraction of the available bandwidth [STRA88a,b,c]. When contention exists, there must be some way to arbitrate between competing tasks in a method that serves the overall purpose of the system. Ranking the tasks via some priority scheme is a reasonable means to provide such arbitration,

although a useful number of priorities is still debatable [PEDE88]. Three observations, however, pervade the use of priorities. First, unless priorities are administered by some central entity (and they almost always are not), they cannot be fair. Second, no communications protocol in use or in development avoids the problem of priority inversion, where a task of lower priority may prevent a task of higher priority from being served. Finally, even if the first two concerns were satisfied, most priority schemes are static, whereas it is more general to consider tasks whose relative importance changes over time according to the changes in the environment.

In this paper we briefly survey the literature on requirements for real-time distributed systems. The literature does not agree, of course, on a canonical set of requirements; we provide this survey to show that there is little agreement in the field about such requirements, and yet some underlying tenets exist, such as the need to be flexible and robust.

We continue by building our own abstraction, the importance abstraction, and relate requirements of a communications subsystem (including real-time requirements) to this abstraction. The importance abstraction is based on the principle that all tasks in a system contribute to the overall system goal, and importance is the measure of their contribution. This is a starting place, where we can make absurd assumptions. It does, however, help us to understand the system concepts and define the roles and responsibilities of the various components of the system and subsystems, especially the communications subsystem.

We then state the goal of this research, which is to find a scheme that can provide a flexible and potent discrimination policy, yet is possible and appropriate for a communications subsystem. We examine typical communications subsystems for what can and cannot be expected from such a policy. This investigation yields an interesting list of open questions and research directions.

Finally, we examine some schemes and ideas as possible solutions to the problem of making a communications subsystem responsive to real-time and other special needs. We find we are stymied by the need for such a scheme to be powerful and yet implementable. We felt it instructive to include "first tries" at offering a scheme, since they have helped us to realize that this problem is genuinely hard.

2. Requirements for a Real-Time Communications Subsystem

It is important to remember that the communications subsytem is not responsible for the attribute *real-time*; rather it is responsive to the needs of the system, and that includes the special needs of a real-time system. What are the special needs of a real-time system? This question has no universal answer; the term *real-time* is vague and ambiguous. Given the emphasis on designing a communications subsystem to support real-time systems, however, the question deserves discussion.

A real-time distributed system consists of several applications that are distributed among different processing nodes connected via a common network. The applications may be executing in parallel subject to both precedence and timing constraints. The communications subsystem has responsibility to provide the services necessary for a timely response to changes in the application's environment. It must provide sufficient functionality and performance to be useful to applications which require time-constrained communication, allowing the application to effectively handle error conditions without the communication system inducing errors of its own ([STRA88d]).

There are three major requirements of distributed real-time processing on a local area network environment [ZNAT87]. *Robustness* is the combination of reliability, availability, and dependability requirements, and reflects the degree of system insensitivity to errors and

misinformation. *Flexibility* relates to the ease of designing and structuring a network that can support real-time processing. Finally, *timing requirements* are the timing guarantees demanded of the network by any given station attempting to access the channel.

The real-time communication system must be efficient and reliable. It must be performance oriented in order to provide a very short response time while ensuring that network errors are detected and are recovered from without involving the network users. There is a trade off: as network error detection and recovery protocols provide more robust and hence more reliable service, the cost of executing these protocols on each protocol data unit increases, degrading performance. Thus it is desirable for a real-time communication system to implement only the necessary communication services. This is a minimum according to the required quality of service.

A major requirement of real-time applications is fault tolerance [HWAN87]. Generally, real-time applications produce specific network load conditions and traffic patterns; the load is relatively light and the traffic pattern varies slightly from predictable behavior [STOI88]. Errors within the application, however, must be detected, confined, and corrected using fault tolerance techniques built into the application. These fault tolerance techniques may cause unpredictable sudden bursts of high priority communication, are called *alarms*. An alarm message must not be stopped by flow control. This sudden peak in load due to alarms is called an *alarm avalanche*. Unless the communications subsystem can meet the deadlines of each of these alarms during an alarm avalanche the error condition will worsen, and more faults will be introduced. It is not sufficient for a real-time system to provide adequate performance during normal activity; the system must actively influence the ability of the fault tolerance techniques to confine and correct the faults before others compound. Real-time system designers must anticipate these message bursts, providing sufficiently robust transfer capacity for all network nodes under all load conditions. This requires high throughput and predictable message delays.

Simply increasing the transmission bit rate cannot relieve congestion caused by burst traffic [MIRA83].

End-to-end resource allocation is also necessary to guarantee response times or throughput. Typically, this is accomplished with end-to-end flow control using windows and credits. However, this method only guarantees that buffer space will be available when messages arrive. Le Lann, in [LELA85], identifies at least two other types of resources that must be allocated, CPU cycles and I/O capacity. In fact, buffers have become a cheap and readily available resource, whereas most LANs are bound by the CPU and system busses.

Connections allocate resources upon creation. However, this allocation may be too costly to maintain if the connection is scarcely used or if there are many connections being supported. The robustness of a communications subsystem must not adversely effect the cost of maintaining the system, whether that cost is measured in resources or message delay.

The most notable and fundamental principle of a real-time communication system is that it must ensure that a message is delivered to its destination before the deadline. Its inability to do so is considered a failure. It may be both useless and dangerous to deliver a message past its deadline, as its significance is no longer a factor and its presence may confuse or overwrite data that did meet its deadline. The mechanisms used for meeting deadlines are priorities and multilevel message scheduling.

Latency control is seen as another requirement. Messages handled at all layers of protocols should be scheduled according to specific algorithms [LELA85]. Establishing finite upper bounds for access delay only solves part of the general problem. Finite upper bounds must be guaranteed for all service times at all layers for given conditions of utilization. Also, a cost function must be minimized when these bounds are exceeded. Such cost functions for a real-time system are the number of messages which miss their deadlines or the average or

maximum message lateness over a given time interval. Deterministic message scheduling algorithms are needed to enforce the guarantee of finite upper bounds as well as ensure that the least costly messages are discarded on overload or in abnormal situations.

3. The Importance Abstraction

Importance is an abstract concept which ranks activities according to how much they contribute to the overall system goal. Within a system there are several subsystems, such as the communications subsystem. Within each subsystem are an arbitrary number of tasks requiring attention. These tasks are created by application processes within the system and are serviced by the appropriate subsystem. For the communications subsystem, these tasks are messages. A subsystem can be viewed as a server, where the application processes within the system are the clients which submit the tasks to the subsystem. In this section we present the importance abstraction, briefly as a characteristic of the general client/server model, then specifically as it applies to the communications subsystem. There are some aspects of the communications subsystem which make the importance abstraction more tractable; we will try to point out when the importance abstraction can not be generalized easily.

Informally, the system is doing the best it can to accomplish the system goal if each subsystem is also doing the best it can to service the tasks it has at hand according to which task will contribute the most toward the system goal. Somehow the subsystem must determine which task is the most important at each instant in time and service that task. The importance of a task is relative to the importance of all other tasks competing for the service within the subsystem, and the task with the greatest level of importance may change as the importance levels of the various tasks change. Therefore, for a subsystem, viewed as a server or a series of servers, to do the best it possibly can given a set of tasks and their relative importance levels, instantaneous preemption is necessary. Ideally, this preemption should save as much work

already done as possible. Thus we consider a client/server model with the following characteristics: (1) each task has some function of time associated with it which describes the task's importance over its lifetime and whose value is always known to the server; (2) the client must provide the enough information to the server to determine the task's importance over its lifetime; (3) at any point in time, all tasks are well-ordered according to their importance, and a *most important* task may be identified and served; and (4) the server is able to preempt tasks instantaneously without loss of work already accomplished. Thus a subsystem that can provide these characteristics will have the further characteristic that it will be doing the best that it is possible to do, according to what is globally considered important.

3.1. Importance Function Definition

Since the various subsystems are simply responsible for servicing the tasks submitted to them by the application processes, the subsystems must be given some mechanism for determining the order of the tasks to service. Within a client/server model, this is known as *scheduling*. Jensen *et al.* [JENS85] define a scheduling policy for operating systems by noting that within computer systems there is a value which varies with time associated with the completion of a process (an operating system's task). This value function, $V(t)$, is the value to the system for completing that process at time t. The goal of the scheduler is to optimize the cumulative value over the lifetime of the system, so that scheduling decisions are made based on what process can contribute the most value.

As a process's value function may increase and decrease with time, scheduling the current highest-valued process (greedy solution) often does not result in the optimal cumulative value. There are processes for which it is beneficial to "hold" them inactive until some time when the values associated with completing these processes have become optimal. This is not typically characteristic of a communications subsystem. Generally, no message is held once inside of the

communications subsystem; it is delivered (completed) as soon as possible. Therefore, a communications subsystem which does the work at hand until no work is left will likely not produce an optimal value, since messages will not be held until their value functions have reached a maximum.

Rather than a function that provides the value of completing a message at some time, the application process should submit some function associated with the message which will describe the message's *importance value* over the lifetime of the message. This importance function is different from Jensen's value function in the following way. Instead of trying to optimize the *value upon completion* of a message, the communications subsystem is using the importance value at every moment in the lifetime of the message to rank that message among other messages, so that it can be working on the *most important* message at all times. The function which describes the message's importance is based on some set of system and environment parameters and maps these into any well-ordered set, for example the integers. Since the system parameters are changing with time, every importance function, $I(t)$, is at least implicitly dependent on time.

There are variables implicit to the system which are measures of conditions which exist within the system. These variables trace the changes in the system according to outside influences or internal activities. An importance function may need to be responsive to these changes, since a message's importance may increase or decrease as these conditions on or within the system change. These system variables must be parameters to the importance functions, and are therefore called system parameters. However, the importance function may not use a particular parameter; in fact, we shall present importance functions that use none, some, and many system parameters.

3.2. Importance Function Issues

Now that we have defined a class of functions that maps a message's importance to the system into a well-ordered set, we raise some issues about this class. There are varying degrees to which an importance function may need to be accurate. As accuracy decreases, generally the complexity of the scheme increases. These schemes may be classified into *types* of importance functions. Furthermore, various applications have differing needs of a discrimination scheme; most do not need a complex scheme but a few require one. We offer a taxonomy of application's needs. Also, the more the importance function can describe, the greater the need to calibrate the function so that its values are comparable within the well-ordered set. This is called *homogenization*. Finally, it is observed that the importance function should be evaluated infinitely often to ensure an accurate ranking of messages. Since this is unrealistic, decision points must be assigned. The degree to which a communications subsystem can provide evaluation of the importance function infinitely often is called *granularity*.

3.2.1. Taxonomy of Types of Importance Functions

The taxonomy of possible types of importance functions is laid out in the tree of Figure 1. Generally, an importance function may actually change with time or may maintain the same value as time elapses. The former are called *dynamic* schemes, the latter, *static*. Static schemes are first broken into two cases: the very special case of one priority for all tasks (that is, no discriminating scheme at all) and multi-level static priorities, where $I(t) = n$ or $I(t) = 2^n$. The multi-level schemes are further divided into the trivial scheme with only two classes of tasks, a scheme with a small enough number of levels of importance that a human being could understand and keep the distinctions between levels in his head, a scheme where a human being with the aid of a handbook could look up the distinctions between levels of importance but could not keep them in his head, and a scheme where the number of levels is so high that they

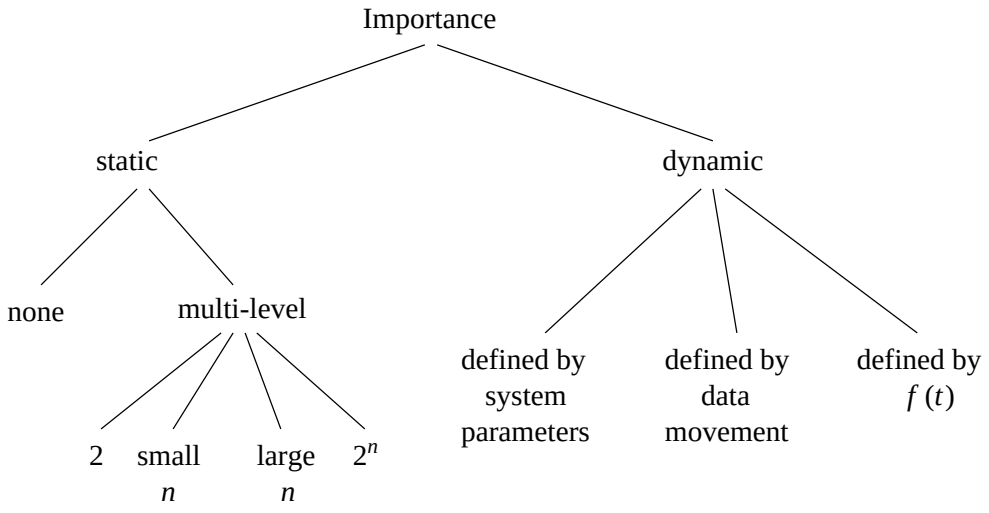

Figure 1 — Taxonomy of Types of Importance Functions

can only be meaningfully assigned by an algorithm and not by a human being. Dynamic importance functions could be defined as either explicit functions of time or by some set of system parameters, in which case importance implicitly depends on time. Or, importance can be based on data movement as in the TCP/IP hop-counts, which are decremented as a packet is handled by a router.

The most general importance function is one that depends on all of the system parameters. The system has many such parameters, like load, congestion at particular stations, level of criticality, and time. Each system parameter may itself be considered a function of time: load and congestion change over periods of time, criticality depends on what is happening at a certain time (such as battle conditions), and time certainly changes. Therefore, the importance function can be described on a planar graph with time as the dependent variable. Unfortunately, since there is no way of knowing what the system parameters will be in the future, there is little

hope of graphing the importance function into the future. (It should be noted that time is a system parameter which can be predicted, so an importance function which depends solely on the current time (such as deadline scheduling) has the property that it may be graphed into the future.)

3.2.2. Taxonomy of Applications

Applications have differing needs for a discrimination scheme. It is desirable to be responsive to all types of applications. Figure 2 shows a pyramid of discrimination schemes according to application's needs. At the top of the pyramid sits a scheme suitable for time dependent traffic, the *deadline traffic scheme*, which provides the most flexible and most complex method of assigning priorities. Only a tiny fraction of all applications, namely real-

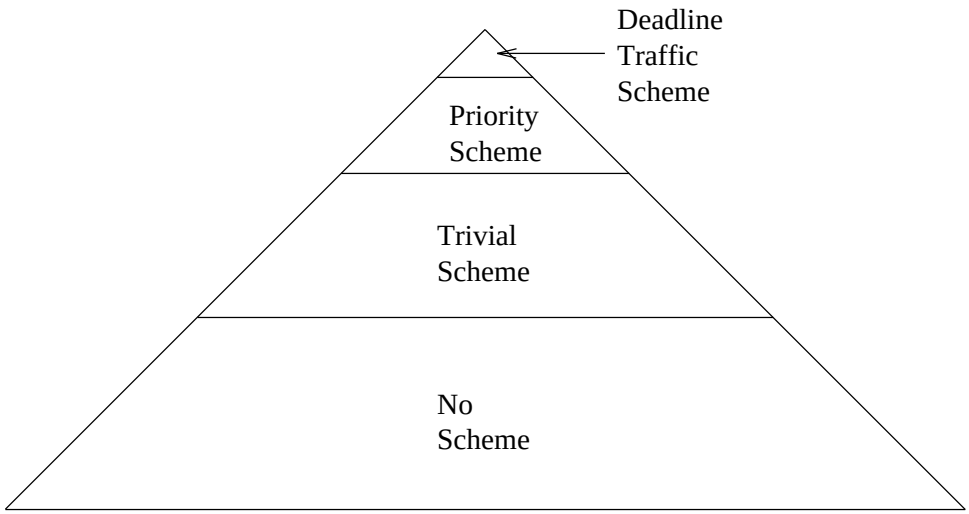

Figure 2 — The Pyramid of Discrimination Schemes

time applications, will need the full power offered by this scheme. A somewhat larger number of applications will desire a *k*-level priority space. This class of applications will be said to need simply a *priority scheme*. A still larger set of applications require only a trivial scheme, having *normal* and *extraordinary* tasks to perform. Finally the broadest class of applications, the base of the pyramid, will not deem any discrimination scheme to be worth the effort and will have all their communications work done at the same priority.

3.2.3. Homogenization

Under every circumstance all of the messages in the subsystem must be well-ordered; that is, the ordering implied by the importance function must reflect the actual order of importance of the messages. Consider deadline scheduling where the message requiring delivery the soonest is the most important. Using only the system parameter of time, the set of messages in the system is well-ordered based solely on the difference between the time-to-serve and the current time. However, if there are also messages which have a static priority (a function of the *null* set of system parameters), making accurate decisions regarding which message is the most important becomes a matter of comparing deadlines against static priorities. In order for the importance function to produce a scalar value from the set of system parameters, each of the parameters must be weighted appropriately. In other words, there must be some *homogenizer* which can ensure that all messages, regardless on what set of system parameters they depend, are well-ordered according to which messages can contribute most to the system goal. The question remains: how do messages with different importance functions, and hence different criteria for importance, relate to each other?

One method of trying to simplify the problem is to enumerate all of the clearly useful importance functions. Now the problem becomes one of relating a finite number of functions at the expense of complete generality. The homogenization problem is simplified further by

restricting the system to a small, closed environment. Designers of a closed system may have some idea of the characteristics of the system and can adjust the importance functions accordingly. At this time, this is considered too restrictive, since wide area networks with many gateways and high bandwidth are soon to be a cornerstone of the scientific computing community.

3.2.4. Granularity and Decision Points

For the statement "the communications subsystem is servicing the most important message at all times" to be true, the subsystem must be able to know the values of importance for each of the messages in the system at each point in time. Since time is continuous, the importance function must be evaluated infinitely often so that at the instant when a message's importance value supercedes the current most important message, that message may be preempted and the new most important message may be served. In reality, this is impractical. The degree to which a communications subsystem can adhere to the above statement is called *granularity*. As the granularity becomes more gross, the communications subsystem can ensure the most important message is being serviced at *almost* every point in time.

Thus, the importance abstraction provides a method for considering a system which has the basic principle that it is always trying to do the most important activity according to what contributes most toward the system goal. Unfortunately, the characteristics of the pure importance abstraction are impractical; no server can preempt messages instantaneously without loss of work, nor can an importance function be evaluated infinitely often at no cost for computation. The importance abstraction is the right place to start, however, as long as relaxing the assumptions mentioned provides a close approximation to the pure scheme. For example, the importance functions may be evaluated periodically so that during the period no changes occur in the ordering of messages. This period should be chosen so that the calculated

importance values could be updated faster than the actual importance can change substantially. This period may be no smaller than a bit-time and still be meaningful, since a bit is an atomic entity within computers. Another scheme is to define decision points based on the observation that there are certain times in a message's lifetime when it is possible to reorder the messages, so that a new, most important message may take the place of the message that used to be the most important. These decision points include at least message entrance into the communications subsystem, transmission onto the medium, servicing within gateways, and exiting the communications subsystem.

4. Applying a Discrimination Policy to a Communications Subsystem

The goal of this research is to find a scheme that will provide a discrimination policy that is flexible and potent. It needs to be flexible enough to apply to all types of networks, from coast-to-coast wide area networks to tightly closed control networks. It must apply to all types of traffic as well, from real-time to background communications. It must be consistent at every installation, yet make no assumptions about the environment in which the network resides. In order to suggest a reasonable scheme to be used within a communications subsystem, we must first observe what most classic transport protocols cannot do with respect to the abstract notion of importance. We can then observe what the transport protocol can and should do. We also recognize a set of open questions and discuss their implications.

4.1. What a Communications Subsystem Cannot Know

A communications subsystem built upon a classical transport layer protocol is not endowed with any special characteristics that would make it suitable for a real-time system. It provides the reliable, end-to-end data transfer service to its users as a part of a communications subsystem. However, there are aspects of the system that the transport protocol cannot know

yet ignorance of them will hinder its responsiveness to real-time needs.

(1) Obviously, no protocol can control for bit errors on the medium or packets lost in the network. These errors destroy any latency guarantees that are not probabilistic.

(2) The "system goal", as defined by the system designers, is neither the communications subsystem's business nor responsibility to understand. As part of the communications subsystem, the transport protocol must provide the best service it can based on the requests issued by the application processes.

(3) Likewise, the communications subsystem cannot know what the enclosing environment is, such as whether the area of distribution is wide or local. There are parameters within the transport protocol which can be tuned and adjusted for its various uses, but the protocol itself should remain consistent under any environment. Specifically, the discrimination mechanism must rely on the same principles under any environment as well.

(4) The communications subsystem cannot assign values of importance to the messages it handles. These messages are important only to the application processes which create or use them, and a message's system-wide importance only makes sense to entities which know the system goal. It is the responsibility of the communications subsystem, and the transport protocol specifically, to be responsive to the needs of the application processes by enforcing the discrimination required; it is not the communications subsystem's responsibility to make up the discriminators it must enforce.

(5) Typical transport protocols are designed to work in conjunction with any physical (and MAC) layer protocols which provide the common basic frame transfer services. It is not known to the transport what protocols are providing these services, so it is not known how to cause the lower layers to be responsive to the discrimination scheme that it is trying to enforce. For instance, Ethernet and FDDI are both considered viable lower layer

protocols for use with various transport protocols; however, Ethernet has no medium access priorities and FDDI has infinitely many of them. Yet, the transport protocol cannot know with which protocol it is being used. Furthermore, it cannot know the signaling speed or the propagation delay between stations, or how many routers and/or bridges there are on the network.

4.2. What a Communications Subsystem Can and Should Know

In spite of things that a communications subsystem cannot do, there are some things that it can and should do to provide some form of discrimination among messages according to their importance to the system. The communications subsystem controls several resources for which messages may contend. These resources include buffer space, processing attention, and network access. The most important message should not have to wait on a resource, since that would imply that a less important message is preventing access to that resource. This is called *priority inversion*. The subsystem should periodically order activities according to which activity has the highest importance value. At each decision point it should give preferential treatment to the most important activity, where a decision point is an opportunity for reordering activities.

With any scheme for providing discrimination, the hope is that some class of messages will benefit from the shedding of the load of all classes of lesser importance. A message with moderate importance may be impeded by messages of more importance; likewise this message may impede the progress of messages of less importance than itself. Degraded service of lesser important messages due to this impedance is the price for providing preferential treatment of the more important messages. The most important message should be the least impeded. Furthermore, impedance on the most important message should be only slightly more than the impedance of a message on a zero-loaded network without any discrimination scheme at all.

Proper management of system resources within the realm of a communications subsystem means avoiding priority inversion and reducing the impedance of the more important messages. One way to ensure that the subsystem has the power to accomplish these objectives is to allow the preemption of less important tasks in favor of more important ones. Preemption is a powerful and costly tool: powerful in that the communications subsystem can take away a resource *at any time* in order to give it to a more important task than the current one, but costly in that when a task is preempted either work will be lost or there will be a good deal of overhead in saving the state of the work. Hence the use of preemption must be sparing else the performance penalty will cancel all advantage.

However the discrimination scheme is implemented within the communications subsystem, a doctrine concerning such schemes must still hold: there should be varying degrees of preciseness, and the price for such a scheme should increase only with the increased precision. As the scheme becomes more precise about which message is the most important, the cost of determining this message will increase. Having no scheme is to have a scheme that does nothing; this scheme provides no discrimination among messages but should provide the highest overall system throughput, since no load of messages are being shed. There should be no overhead due to a discrimination scheme if one does not use the discrimination scheme. Four tiers of precision are useful: none, trivial, static, and precise. No scheme provides only a normal data transfer service, it is up to the application process to discriminate. A trivial scheme is a binary division, perhaps like ISO Transport Protocol Class 4's normal and expedited services. The static scheme provides a range of priorities, which, once set, are not changeable. Finally, the precise scheme provides a dynamic priority which is raised and lowered by the measure of importance that message has toward the system. These four schemes should be able to coexist. Clearly, the more precise the scheme, the more it will cost to calculate which message is the most important. Herein lies a delicate trade-off: if the ability to provide very a

precise method of preferring one message over a set of others costs more than the benefit gained by the preferred message, the scheme only hinders the system performance. A discrimination scheme is an investment which must show tangible benefits to the class of messages whose performance it wishes to improve.

As suggested in Figure 3, no discrimination scheme at all results in a first come first serve (FCFS) service discipline. Any attempt to order messages into importance classes is equivalent to having multiple queues which the server must drain. Since having no scheme requires less effort, it must be that the throughput for the FCFS system is higher than the system with discrimination. As a doctor's first rule is to do no harm, so a protocol designer must be sure that any proposed discrimination scheme does not result, due to the computational overhead of determining the importance of each message, in a lower latency for the class of preferred messages than what would have resulted from an unintelligent FCFS discipline.

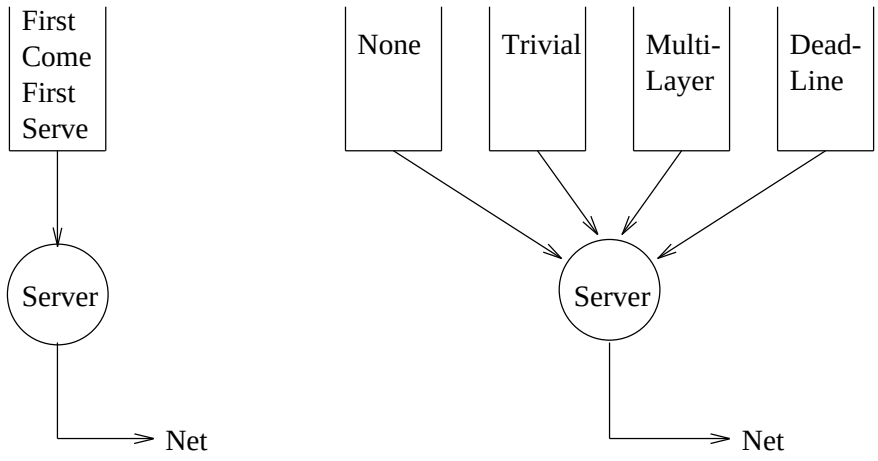

Figure 3 — Unintelligent *vs.* Discriminant Schemes

The communications subsystem should provide a good interface to the user, regardless of how precise the discrimination scheme may be. The interface should be simple, low overhead, consistent, orthogonal, regular, and clear. The scheme should be able to operate unambiguously from the parameters which are provided to it by the application process.

Regardless of the scheme used, a message placed into the communications subsystem should be delivered as soon as possible, according to its relative importance with the rest of the messages contending for the attention of the subsystem.

4.3. Open Questions

There are many questions which must be answered before the definition of a priority field can be made completely.

(1) What are all of the decision points in a communications subsystem, where activities may be reordered if warranted by the discrimination scheme? Since a transport layer protocol cannot control the layers below it, its only realm of influence is the processing it performs on the message itself.

(2) The interface to the communications subsystem must convey enough information to allow the subsystem to be responsive to the relative importance values of the messages. What parameters must be included in this interface? How will the scheme use them? Also, how will the transport layer protocol, once it knows a message's importance, convey that importance to the underlying services, without knowing what they are?

(3) Real-time applications are by definition constrained by time. How necessary is it to have a time-based discrimination scheme? If it is necessary, common time must be distributed to all network nodes (not an easy task), and it must be an order of magnitude more accurate than is necessary. Algorithms for time distribution exist ([GORA]), but they are

complicated and hinder efficiency. The communications subsystem cannot get the system time through system calls; this is surely too costly. Perhaps the subsystem should have its own time of day clock. Even so, accuracy is a major issue, and a time-based scheme would be gross at best. Real-time application processes may have deadlines and time constraints, but it is not clear that the communications subsystem really can enforce deadlines. It can give preferential treatment according to a deadline and some other weighting factors, but the deadline time itself may not actually need to be included in the priority field.

For the purposes of this discussion a deadline is a time by which the receiving operating system must be notified that a message has arrived. This definition is reasonable since the communications subsystem can do no more than deliver a message to an exit queue. After that point the message leaves the communications subsystem's sphere of influence.

Even assuming that having an accurate system clock is a solved problem, the use of deadlines remains problematic. A deadline handed to the communications subsystem by an application process is not meaningful for calculating importance. This raw deadline contains no information about the time that it will take the message to be delivered and hence does not accurately reflect the urgency of the message's delivery from the communications subsystem's point of view. To make a deadline, *D*, useful in ascertaining importance, the following times must be known (or estimated): time required for the subsystem to seize the necessary resources (t_{SR}), time to gain network access (t_{NA}), time to transmit (t_{TR}), propagation time (t_P), time in routers and gateways (t_{RG}), and time in receivers (t_{RVR}). If these times are subtracted from the deadline, one gets the "latest time to transmit", t_{TRANS}, which represents a "true deadline" from the communications subsystem's perspective.

$$t_{TRANS} = D - t_{SR} - t_{NA} - t_{TR} - t_P - t_{RG} - t_{RVR} \quad (4.1)$$

A plausible discrimination scheme then would be to service messages by an earliest transmit time first discipline. Unfortunately, many of the variables in Equation 4.1 are unknown, are hard to measure, or have a very high variance. There is no way to know or predict with accuracy how long a message might spend in routers and gateways or in an entrance queue waiting for access to the network. The only time that is easily computed is the time to transmit which is simply the size of a message divided by the capacity of the network. But this fact is of little use since t_{TR} variable is one of the least significant terms in Equation 4.1. Moreover, even if the intractable problem of computing t_{TRANS} were solved, an underlying assumption in earliest transmit time first is that two messages with the same deadline are equally important, which may not be the case.

(4) It is not clear that a message will be in a communications subsystem long enough for its importance value to change. This surely depends on the environment; wide area networks with several gateways may impose a large latency on messages. If all messages can be delivered before any appreciable change occurs in their importance values, then a dynamic scheme may not be necessary. It is probably true that some scheme is necessary to resolve contention, whether this is dynamic is an open question.

(5) What assertions and guarantees fall apart during the presence of errors in the communications subsystem? Transport layer protocols provide a reliable service, which implies some need for acknowledgements or negative acknowledgements. During errors, latency may be as much as two orders of magnitude larger than during error-free operation. Resource remain tied up while the transport layer tries to recover. Under these circumstances, it is clear that a static scheme is all that is necessary? What importance values should acknowledgements and retransmitted data carry?

These are a few of the observations and questions our research has lead us to discover.

5. Toward A Workable Discrimination Policy

Each task in a subsystem must have some method of encoding the importance function within it so that the server can choose the most important task to serve. As the system parameters change, the value of the importance function changes to respond to the new environment. The server can evaluate the importance function as it needs to know a task's importance value; ideally this evaluation would be done infinitely often with no cost so that the instant one task becomes the most important task over the previously most important task, the server can preempt the old task and serve the new one.

Admittedly, this scheme is unrealistic for any subsystem. It is nonetheless the place to start since the assumptions which are made about the system can be relaxed later. This scenario represents the best that can happen because the system is in essence being driven by an oracle which can properly assign importance functions to each task and thus determine how the server can be most responsive to the system's overall goal.

First and foremost among the difficulties with the abstraction of importance is how to assign importance functions that will accurately describe a task's importance during its lifetime. In order to do this the system designer must know *a priori* what other tasks are possible at every moment during the task's lifetime, what their importance functions may be, and furthermore must ensure that some task does not preempt a task which is really more important or is not preempted by a task which is really less important due to an improperly assigned importance function.

5.1. Task Types and Useful Curves

While it is not clear that a canonical list of task *types* exists, one possibility for the communications subsystem includes: deadline, dynamic priority, static priority, and no priority scheme at all. Each task's importance is derived from what is important to it. For example, if it is important for a task to be completed before a deadline, the importance function will depend upon the system parameter of time and the task's deadline. Other parameters, such as whether the task has a hard or soft deadline, will help rank it among other deadline traffic, and in fact, among all other traffic.

Although the importance abstraction is powerful enough to provide a system with the mechanism necessary to impose a ranking among tasks competing for the server, without a canonical list of the task types it is virtually impossible to write down a function which will adequately describe a task's importance over its lifetime. The value of an importance function $I(t)$ changes as the set of system parameters upon which it is based changes. How the value changes depends on the nature of the importance function. There are many possibilities to consider: should $I(t)$ be linear, stepwise, or exponential? Is $I(t)$ decreasing, increasing, or monotonic? Is $I(t)$ everywhere defined? Based on our own experience, experience reported in the literature, and discussions with several system designers, we have attempted to draw several *shapes* of curves that might be useful importance function shapes.

The simplest to draw and to understand is the static priority scheme, Figure t, where the application process provides the task with a number, or fixed importance level, which it will keep during its entire lifetime. The origin of the graph, t_0, is the birth of the task, or when the task enters the subsystem. Figure 5 shows a hard deadline example where the importance increases exponentially to infinity as the critical time approaches. If the task is not satisfied by the critical time, its importance value becomes negative, indicating that it is very *un*important.

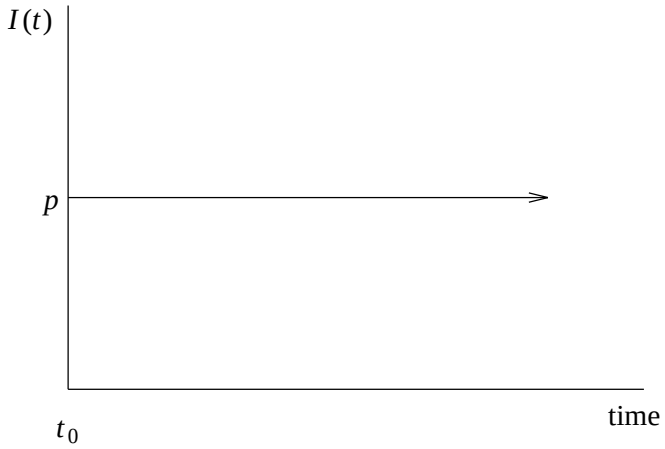

Figure 4 — Static Priority

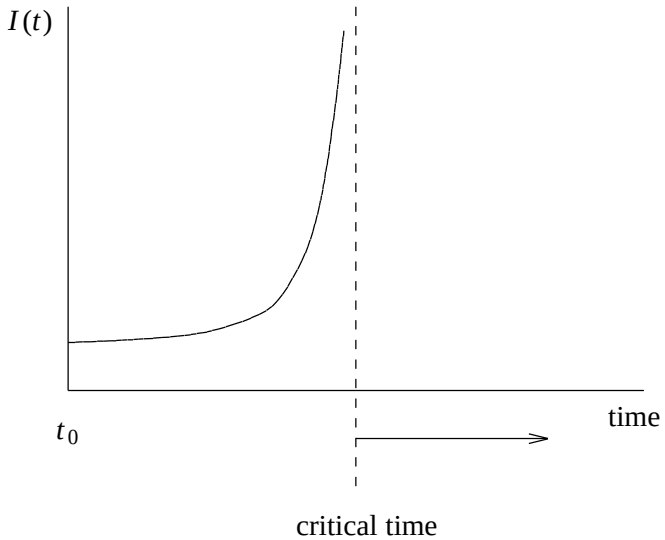

Figure 5 — Hard Deadline

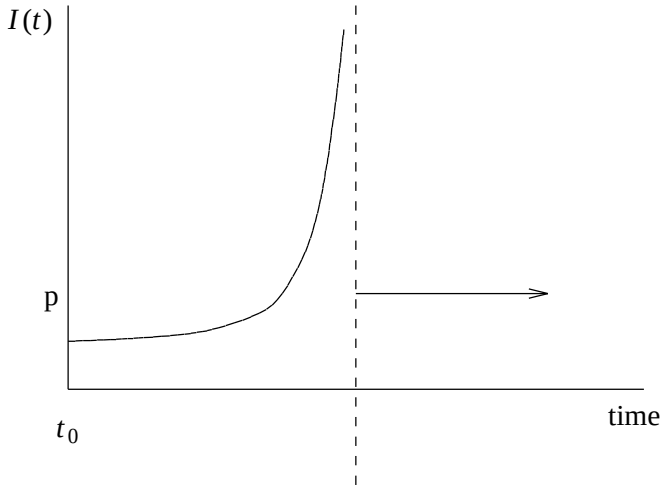

Figure 6 — Soft Deadline

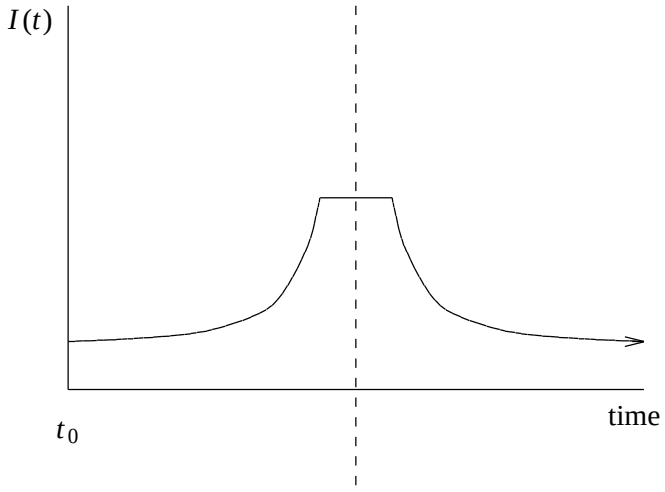

Figure 7 — Window of Opportunity

Figure 6 shows the same importance function before the critical time is reached, only now the task retains some importance even if the critical time passes, as might a task with a soft deadline. Figure 7 shows an importance function of a task that has a "window of opportunity", where the task's greatest importance value occurs over an interval of time centered on the critical time. Notice that in this case the importance function is symmetrical about the critical time. This indicates that the task is as important to satisfy some time before the critical time as it is to satisfy the same amount of time after the critical time. Of course, there are many variations on these themes; the point was to show some useful shapes of curves which may be used for the importance functions of certain types of tasks.

5.2. Ideas on Homogenization

Another problem with the importance abstraction is that the values of the importance function must somehow relate to one another in order to impose a ranking. Thus, if one importance value is higher than another, it is defined to be more important. However, in order to ensure that all importance functions, and there can be infinitely many of them, are mapped onto a planar space in a manner which is consistent, each importance function would have to be carefully calibrated. For example, consider an importance function of one task that is based on a deadline such that the importance of the task is exponentially increasing as the deadline approaches, such as Figure 5. Also consider the importance function of another task which is static, like Figure 4. How are these tasks related? At what point should the deadline task become more important than the task with static importance? What parameters must be included so that the two functions share the same planar space? Each possible task's importance function would have to be compared to each other's importance function, which is combinatorially explosive.

The following is an idea which can simplify matters without compromising the integrity of the importance scheme. It is not clear at this time whether the premise is completely correct. The premise is: all tasks which are dependent on deadlines are *inherently more important* than any other tasks. Thus, all tasks can be split into the two categories: deadline and priority. Priority tasks are tasks whose importance function is either a constant (static priority) or implicitly dependent on time by being explicitly dependent on some system parameter which is dependent on time (dynamic priority). Deadline tasks no longer have to share the same importance space as priority tasks; there is now an importance space for each category of tasks.

In terms of implementation there could be two queues, one that dealt exclusively with deadline tasks, and one that dealt with all other tasks. Within these queues, tasks would be ordered by their individual importance functions; however, all tasks in the deadline queue are satisfied before any task in the other queue is. This clears up the question about how deadline and non-deadline tasks are related — the deadline tasks are *always* more important.

It should be noted that there is at least one important exception. An alarm message is always more important than any other message, except possibly another alarm message. Alarms do not necessarily have deadlines, but would be more important than any deadline message. To handle an alarm by the scheme above, one would have to artificially treat it as a message with a deadline, even though no deadline is appropriate.

5.3. Example Scheme

Another idea has more promise. It is based on the observations that (1) calculation of an arbitrary function to determine the importance of a task may be too expensive, and thus actually hinder rather than help the application processes; (2) the importance function should be evaluated fairly often to catch any changes; and (3) a series of static importance values may

come close enough to approximating a continuous importance function to be usable. The following discussion builds on these observations.

The priority field associated with a message will contain an encoding for three decision times, call them t_1, t_2, t_3, at which the importance may be raised or lowered. (Note the existence of decision times is predicated on a common time reference within the communications subsystem.) Also encoded for each region on the time line will be four integers, p_1, p_2, p_3, and p_4, in the range -1 to some maximum, n, where higher numbers are assumed to represent higher importance values. The priority values for each region of the time line, that is p_1, p_2, p_3 and p_4, may be assigned any value from -1 to n. Figure 8 shows how the time line is divided.

Recall the four classes of priority schemes shown in Figure 2. All can be accommodated in one importance space under our proposed mechanism. Time dependent traffic, or *deadline traffic*, is inherently more important than traffic that is not time dependent. For this reason *deadline traffic* will have the use of all possible priority values, -1 to n. *Priority traffic* will have the values from -1 to $n/2$ available to its importance functions.

The value -1 will represent the *kill* priority, which is a mechanism useful in controlling the lifetime of a task within the communications subsystem. The importance function then can be computed by determining into which region of the time line the current time falls and reading the priority value for that region.

To illustrate, consider Figure 9 where the hard deadline case of Figure 5 is modeled. The deadline (or critical time) can be encoded as time t_3 in the sort field. Initially the importance of the task is small, so the priority may be chosen to be less than $n/2$ to allow the system to service other (possibly non-deadline) tasks. At time t_1 the importance jumps to a higher priority. In particular it jumps to a priority above $n/2$ so that this deadline task is now assured of

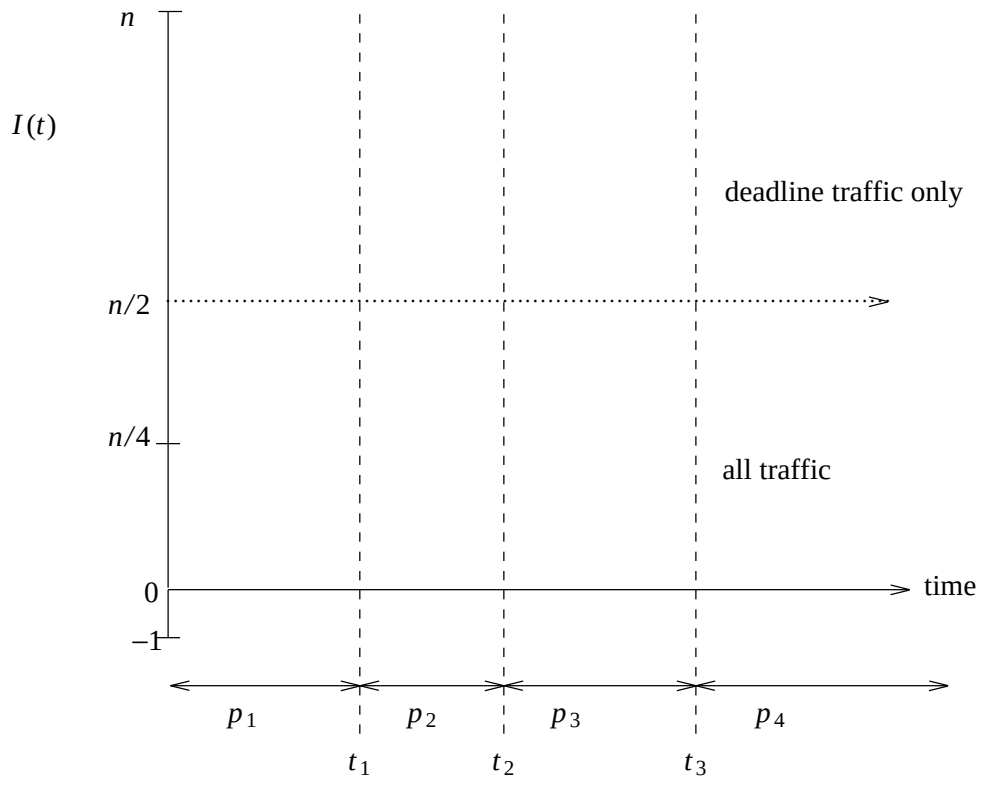

Figure 8 — The Importance Space with Three Decision Times

preempting all traffic in the lower priority schemes. At time t_2 the importance goes still higher since the approaching deadline increases the urgency of performing the task. Finally, since the semantics of a hard deadline application dictate that the task should not be completed if it misses its deadline, after time t_3 the task is given the kill priority. (To model instead the soft deadline of Figure 6, p_4 would be some (presumably low) non-negative priority.)

An important aspect of this scheme is its ability to ensure that during certain time intervals a deadline task does not have to compete with any non-deadline traffic. A deadline task is assumed to be inherently more important as the deadline approaches than any time

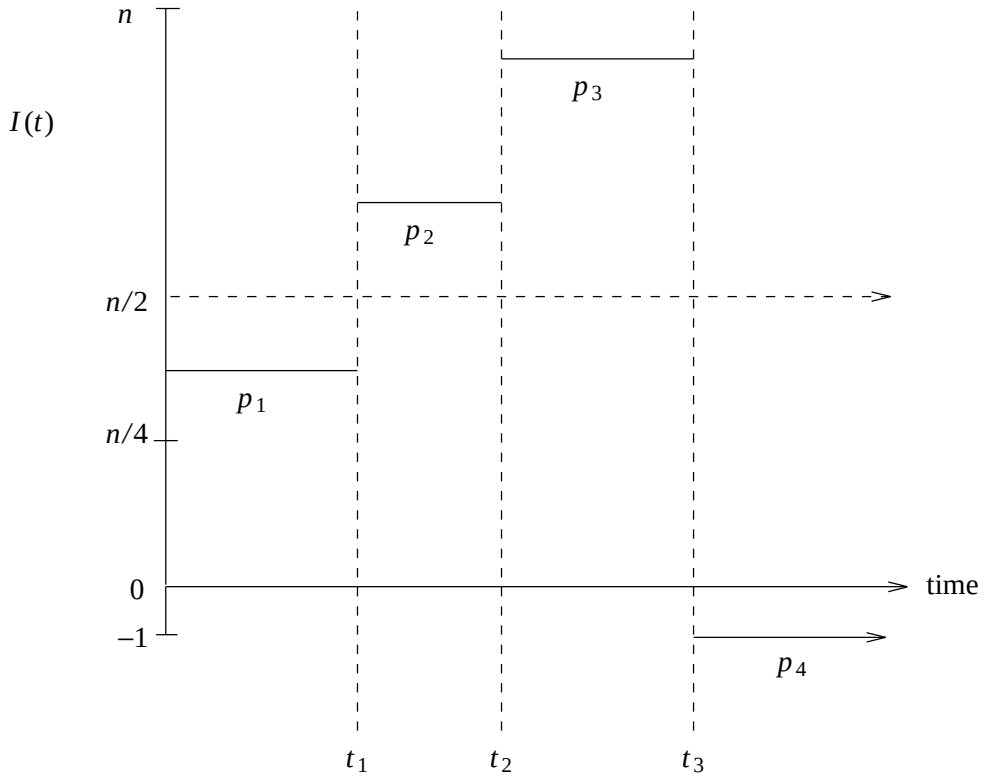

Figure 9 — An Example

independent traffic; hence the highest priority values are reserved for use by deadline traffic only. On the other hand the overall efficiency of the communications subsystem and its fairness are improved by allowing time dependent traffic to be assigned priorities less than $n/2$. A deadline task's importance is indeed low when the deadline is far into the future or, in some cases, after the (soft) deadline has passed.

If desired, the deadline can be encoded as time t_2. Now, when the deadline is passed, a task may have its priority reduced in the interval between time t_2 and t_3 and then be killed after

time t_3. In this way a task that has only an interval of time after its deadline during which its completion would be useful can be accommodated.

An exact deadline means that the task will be most effective if completed precisely at the deadline. This is the "window of opportunity" illustrated in Figure 7. Here the importance function should have p_2 and p_3 as the same (high) priority and p_1 and p_4 as the same (low) priority. In this way the task is most likely to exit the communications subsystem in the interval from time t_1 to time t_2.

Some applications will need only a *k*-level priority scheme for their traffic. For dynamic priorities the importance value of a task may be raised or lowered at each of the decision times t_1, t_2, and t_3. An application achieves static priorities by assigning p_1, p_2, p_3 and p_4 to all have the same value. A common use of p_4 will be to control the lifetime of a task; that is, p_4 may be given the *kill* priority as its value so that a task will not remain in the communications subsystem beyond a certain time, t_3.

Applications requiring only a trivial priority space, that is, one with two priority values, or no scheme at all, need only static priorities. For the trivial scheme *normal* traffic should be mapped to priority 0 and *extraordinary* traffic to some middle value in the importance space of all *priority scheme* traffic, namely *n*/4. If it uses no scheme at all, the application has indicated no concern about discriminating between tasks, and the communications subsystem cannot divine importance on its own. The lowest importance value, 0, is thus the logical value to assign all tasks from these applications.

6. Conclusions

This is an ongoing research project for which the answers to many of the open questions still need thought. We are attempting to make a new communications protocol responsive to

the needs of its users, especially the very constrained needs of real-time systems. Without designing the communications subsystem from the bottom up, endowing each component with the attributes that are required by the importance abstraction, this is a truly difficult task. Yet we still wish to give current transport protocols the mechanism to do the best it can to meet the needs of its users. We are identifying the aspects of these transport protocols and the communications subsystem in which it is a part that are hindering this goal, and those characteristics that are useful in simplifying the task. We have presented here our initial abstraction of importance, how importance can be related to communications subsystems, and some ideas and schemes which probably will not prove to be the ultimate encoding schemes but are useful and instructive toward gaining closure on our goal.

ACKNOWLEDGEMENT

This work was sponsored by the Naval Ocean Systems Center, San Diego, California, and adminstered by the National Institute of Standards and Technology, Gaithersburg, maryland, under contract number 70NANB8H0811.

REFERENCES

GORA86 Gora, W., Herzog, U., and Tripathi, S.K., "Clock Synchronization on the Factory Floor", *Proceedings of the Workshop on Factory Communications*, National Bureau of Standards, pp. 87-108, March 17-18, 1987.

HWAN87 Hwang, K.W., Tsai, W.T., and Thuraisingham, M.B., "Implementing a Real-Time Distributed System on a Local Area Network", (Abstract), *Proceedings of 12th Conference on Local Computer Networks*, pp. 142, October 5-7, 1987.

JENS85 Jensen, E.D., Locke, C.D., Tokuda, H., "A Time-Driven Scheduling Model for Real-Time Operating Systems", *Proceedings of the Real-Time Systems Symposium*, December 3-6, 1985, pp. 112-122.

LELA85 Le Lann, G., "Distributed Real-Time Processing", *Computer Systems for Process Control*, pp. 69-90, 1985.

MIRA83 Mirabella, O., "Real Time Communications in Local Area Network Environment", *Proceedings of MELECON '83*, A1.07 Vol. 1, 1983.

PEDE88 Peden, J.H., and Weaver, A.C., "The Utilization of Priorities on Token Ring Networks", *13th Annual Conference on Local Computer Networks*, Minneapolis, Minnesota, October, 1988.

STOI88 Stoilov, E.I., "Coordination mechanisms in Local Area Networks for Real-Time Applications", *Computer Communications Systems*, IFIP, 1988.

STRA88a Strayer, W.T., and Weaver, A.C., "Performance Measurement of Data Transfer Services in MAP", *IEEE Network*, Vol. 2, No. 3, May 1988.

STRA88b Strayer, W.T., and Weaver, A.C., "Performance Measurements of Motorola's Implementation of MAP", *13th Annual Conference on Local Computer Networks*, Minneapolis, Minnesota, October, 1988.

STRA88c Strayer, W.T., Mitchell, M., and Weaver, A.C., "ISO Protocol Performance Measurements", *ISMM International Symposium Mini and Microcomputers*, Miami Beach, Florida, December 14-16, 1988.

STRA88d Strayer, W.T., Weaver, A.C., "Evaluation of Transport Protocols for Real-Time Communication", *Department of Computer Science Technical Report No. TR-88-*,

ZNAT87 Znati, T., and Ni, L.M., "A Prioritized Multiaccess Protocol for Distributed Real-Time Applications", *Proceedings of the 7th International Conference on Distributed Computing Systems*, pp. 324-331, 1987.